Neural Networks.
Lab 1: Introduction in MatLab and Neural Networks Toolbox

I. Short overview of MatLab *(MATrix LABoratory)* *(http://www.mathworks.com/)*

MatLab is a high-performance language for technical computing which integrates computation, visualization, and programming in an easy-to-use environment. It is an interactive system whose basic data element is an array that does not require dimensioning. Typical applications are in the following domains:

- Math and computation
- Algorithm development
- Data acquisition
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

1. Matrices and vectors

A matrix is a rectangular array of numbers. The vectors are particular matrices containing either only one row or only one column. Similarly, the scalars are 1-by-1 matrices. Almost all operations apply directly on matrices.

1.1. Defining matrices

- *By explicitly specifying the elements*:
  [a11 a12 ... a1n; a21 a22 ... a2n; ... ;am1 am2 ... amn]

Example: A=[1 2 3; 4 5 6]

Remark : The elements of a row are separated by comma or space while the rows are separated by a semicolon.

- *By using built-in functions*: allow the definition of particular matrices

Examples: eye(5) (identity matrix of size 5),
zeros (2,3) (null matrix of size 2 *x* 3),
rand(3,4) (matrix 3x4 with random elements uniformly distributed on [0,1]),
randn(3,4) (matrix 3x4 with random elements distributed according to the standard normal distribution).

- *By loading the elements from a file*: load ('<filename>')

Example: load(matrix.dat) – a variable named matrix is generated; it will contain the values in the file (each line in the file will correspond to a row in the matrix).

Remark. The command load can be also used to load the workspace previously saved in a file having a filename with the extension ".mat". The saving command is save.

- *By concatenating matrices of compatible sizes*: if A is a matrix of size *mxn* and B is a matrix of size *mxk* then c=[A B] will be a matrix of size *mx*(*n* + *k*) obtained by concatenating the columns of A with the columns of B. By concatenating the transposes A' with B' as in d=[a';b'] one obtains a matrix of size (*n*+*k*)*xm*.

Remark. The operator ' applied to a matrix generates the transpose of that matrix.

1.2. ***Accessing the elements of a matrix :*** matrix(row index, column index)

*Example:*change the value of an element: *A(i,j)=expression*

1.3. ***Ranges and submatrices:***

- m:n (the set of all values larger or equal to m and smaller or equal to n),
- a:h:b (the set of values {a,a+h,a+2h,…,b}),
- A(:,j) (column j of matrix A),
- A(i,:) (row i of matrix A; i.e. A(1:3,:) denotes the first three rows of A).
- A(i1:i2,j1:j2) (the submatrix containing all elements having row indices between i1 and i2 and column indices between j1 and j2)

Remarks.

1. The step value, h, used in specifying ranges of real values can be positive or negative.

2. To find the size (number of rows and number of columns) of a matrix one can use the function size. When called for a matrix this function returns a pair of values: [number of rows, number of columns], i.e. [lin col]=size(A).

3. The empty range is specified as []. A matrix can be easily reorganized by eliminating rows or columns. For instance, by a(1,:)=[] the first line of the matrix is deleted while by a(:,2)=[] the second column of the matrix is deleted..

1.4. ***Operations on matrices:*** addition (+), subtraction (/), multiplication (*), transposing (‘), power (^), inverse of matrix A (inv(A)), compute eigenvalues of matrix A (eig(A)), finding x which satisfies A*x=b (x=A\ b), finding x which satisfies x*A=b (x=A/b). For operations at the level of elements one can use the dot-operators .*(for multiplication), .^ (for computing powers), ./ (for division).

1.5. ***Relational and logical operators:*** equal (==), different (~ =), strictly less than (<), less than or equal (<=), strictly greater than (>), greater than or equal (>=), negation (~), disjunction(|), conjunction (&).

1.6. Variables and assignment: the variables can be used without prior declaration; they can be matrices, vectors of scalars of one of the types: integer, real, complex (the symbols used in specifying complex values are i or j). To visualize the list with variables used in the current session one can use the command who or just analyze the content of the window Workspace. The command used to reset the variables is: clear <variable name>

1.7. Bult-in constants: pi (π), i (constant used to specify complex values), inf (symbol for infinity), NaN (undefined value, i.e. 0/0), eps (floating point accuracy: 2^{-62}).

1.8. ***Built-in functions:*** sin, cos, tan, asin, acos, atan, exp, log (natural logarithm), abs (absolute value), sqrt, sign (signum function), round, floor, ceil, rem (remainder of the division), max, min, sum, prod, mean, std (standard deviation), median, sort (increasing sorting), find (find the indices of all elements in a vector satisfying a given property).

Remark. When the functions max, min, sum, mean etc. are applied to a matrix they are separately applied to each column of the matrix leading to a row of results.
.
Example: to obtain the smallest element of a matrix: min(min(A))

1.9 Strings, I/O operations

The strings are specified by using ' , i.e. 'this is a string'.

To print the values of variables one can use disp which can be used with only one parameter (example: disp('Result:');disp(res)). The error messages can be printed by using error which also interrupts the execution of the sequence of commands. The input data can be read by using input (example: var=input ('Input value:'))

2. Programming in MatLab

2.1. ***Scripts and functions.*** MatLab allows to group several commands in a file (script file) or to define and save functions in files in order to call them from the command window. In both cases the files can be created using the MatLab editor and their name should be *.m.

Script files: consist of MatLab commands which are executed when the name of the file is specified in the command window (or when the script is activated from the editor). All variables used in the script are global (any change of their values is reflected in the global workspace).

Function files: are used to describe small programs. The file can be edited using the editor and the name should be the same as the name of the function: functionname.m. The variables used in a function are local ones, except for those which are declared to be public (by specifying their names in the function header) and which are used to collect the results provided by the function. The header of the function is:

```
function[res1,res2,…resn]=<namefunction>(par1,par2,…,parm)
```

After the header is specified the function body which is terminated by end. A defined function is called by specifying its name followed by the list of parameters values. A m-file can contain several functions but only the function having the same name as the file is visible from the command window.

Example: function to compute the mean and the standard deviation for a vector (matrix with one column)

```
function [amean, stddev]=statistics(x)
m=size(x,1); % find the number of elements in x
amean=sum(x)/m;
stddev=sqrt(sum(x.^2)/m-amean^2);
end            % the terminator is optional
```

After the function is defined it can be called as: `statistics([1;3;4;5;4])`

2.2. *Flow control*

Conditional statements:

```
if <condition>
   <statements>
else
   <statements>
end
```

The condition is usually a logical expression. If a branch contains several statements then they should be placed on different lines or, if they are on the same line, they should be separated by commas. Nested if can be described as:

```
if <condition>
  <statements>
elseif
  < statements>
else
  < statements>
end
```

Example. Function which compute the signum:

```
function [s]=sign(x)
if x>0
  s=1
elseif x==0
  s=0
else s=-1
end
```

In the case of several branches one can use the switch:

```
switch <expression>
  case <choice1>, <statement1>
  case <choice2>, <statement2>
  ...
  otherwise, <statement>
end
```

Example:

```
arg=input('Value in the set {1,2,3} =')
switch arg
case 1, disp 'one'
case 2, disp 'two'
case 3, disp 'three'
end
```

In Matlab one can use one of the following looping statements:

```
for <variable> = <range>
    <statement>
     ...
    <statement>
end

while <condition>
   <statement>
    ...
    < statement >
end
```

Examples.

1. `for i=1:10, i, end` just print all values of i

2. `while(n), disp(rem(n,2)),n=floor(n/2);, end` prints the binary digits of n (in reversed order)

3. Graphics

2D: to construct the graph of a function f:[a,b]->R one have to define the discretization of the input interval and then use the plot command:

```
x=a:h:b;
plot(x,f(x))
```

where *h* is the discretization step. To plot together several graphs one can specify:

```
x=a:h:b;
plot(x,f(x),x,g(x),x,h(x))
```

The style can be specified by using several options which are extra parameters of plot:

- color: is specified by a letter identifying the color (r-red, b-blue, g-green, y- yellow etc.),
- line type: is specified by a symbol: - (continuous line, - - (dashed line), : (dotted line)

- markers for points on the graph: +, o, diamond, square etc

  Example: `plot(x,f(x),' r-o')`

Example.

```
x=0:0.1:2*pi; plot(x,sin(x),x,sin(x/2),x,sin(2*x));
xlabel('x');ylabel('sinus');
legend('sin(x)','sin(x/2)','sin(2x)');
```

To visualize several graphs on different regions of the same window one can use the command `subplot(m,n,i)` where m and n denotes the sizes of the table containing the subwindows and i is the number of the subwindow (counted row by row).

Example.

```
 x=0:0.1:4*pi;
subplot(1,3,1);plot(x,sin(x));xlabel('x');ylabel('sin(x)');
subplot(1,3,2);plot(x,sin(x/2));xlabel('x');ylabel('sin(x/2)');
subplot(1,3,3);plot(x,sin(2*x));xlabel('x');ylabel('sin(2*x)');
```

3D: To plot a surface (corresponding to a function f defined on $[a; b]\ x[c; d]$) there are several steps to follow:

- define a grid of the domain (with given discretization steps):
  `[x,y]=meshgrid(a:hx:b,c:hy:d)`
  x and y will be matrices containing the values of coordinates x and y respectively

- compute the values of f on the grid nodes `z=f(x,y)`

- plot the surface by using `mesh(x,y,z)` or `surf(x,y,z)`

*Example.*Plot $f\,(x,y)=x^2+y^2$ on $[-2, 2]\ x\ [-2; 2]$ using hx=hy=0.1:

```
[x, y]=meshgrid(-2:0.1:2);
z=sqrt(x.^2+y.^2);
mesh(x,y,z); pause;
surf(x,y,z); pause;
meshc(x,y,z); pause;
contour(x,y,z); pause;
contour3(x,y,z);
```

4. *Other data structures:*

Multidimansional arrays: extend the matrices; need more than two indices to specify the elements.

Example. `a=rand(2,3,4)` – generate an array of 4 matrices, each one having 2 rows and 3 columns. The element on position (i, j, k) is specified as a(i,j,k).

Cell arrays. Allow to group arrays of different sizes. They are specified by using { and } and their elements are identified by indices.

Example. Construction of an array with 3 cells containing matrices of different sizes:
```
a=rand(2,2); b=rand(3,3); c=rand(4,4); d={a b c}
```

The first cell can be specified as: d{1}. The first row of the first cell can be specified as: d{2}(1,:).

Remark. When the cell array d is created the content of matrices a,b, and c is copied in d.

Structures. Are collections of objects of different types. Each object is identified by its name. They are useful to define neural networks architectures.

*Example.*A structure name art can be defined by filling in each field:
```
        art.name='popescu'; art.table=rand(2,2);
```

One can define an array of structures by filling in new elements as in:

```
        art(2)=struct('name','ionescu','table',rand(2,2))
```

II. Neural Network Toolbox

1. It contains a set of MatLab functions which implement architectures and learning algorithms for several types of neural networks. The user can specify the architecture, the activation functions, the connectivity, the learning algorithm. To get a first idea on the facilities offered by NN toolbox just use help nnet (it lists all functions related with neural networks).

There are several demos which can be activated by using nnd.

The toolbox contains both general functions (as sim to simulate a network or adapt and train to train a network) and particular functions for given architectures (as newlin, newff, newrbf etc.).

Any neural network implemented in NN Toolbox is a structured object having as components arrays, functions for simulation and training and control parameters.

Some of the examples illustrating the power of NN Toolbox are

- linear identification in signal processing (applin1),
- time series prediction (applin2) ,
- pattern recognition (appelm1),
- character recognition (appcr1)

Also, there are some applications with GUIs:

- pattern recognition: nprtool
- pattern fitting: nftool
- clustering: nctool

2. Activation functions::

- Heaviside: hardlim
- signum: hardlims
- linear: purelin
- saturated linear (valued in [0,1]): satlin
- saturated linear (valued in [-1,1]): satlins
- sigmoid (valued in (0,1)): logsig
- hyperbolic tangent (valued in (-1,1)): tansig

Exercise 1. Graphs of activation functions.

Hint. x=-5:0.1:5; plot(x,logsig(x)) or by using subplot

```
x=-5:0.1:5;
y1=hardlim(x); y2=satlin(x); y3=logsig(x);
subplot(1,3,1);
plot(x,y1,'r'); % red graph
title('Heaviside'); % title
subplot(1,3,2);
plot(x,y2,'b'); % blue graph
title('Saturated linear [0,1]');
subplot(1,3,3);
plot(x,y3,'g'); % green graph
title('Logistic');
```

Exercise 2. Study of the influence of the activation threshold on the neuron output.

Hint. Plot on the same graph $f(x-p1)$,$f(x-p2)$ ..., where f is an activation function and $p1$, $p2$ ... are values of the threshold.

Exercise 3. Study of the influence of the slope value on the neuron output.

Hint. Plot on the same graph $f(p1*x)$,$f(p2*x)$ ..., where f is an activation function and $p1$, $p2$ ... are values of the slope

Exercise 4. Use nnd2n1 and nnd2n2 to study the behavior of a neuron.